Facilitating personal content management in smart phones

Aaltonen, Antti

CHAPTER 7

META-SEARCHES IN PEER-TO-PEER NETWORKS[5]

7.1 INTRODUCTION

The amount of digitally stored information increases rapidly. The hard disks of today's computer users are filled with thousands and thousands of files in a myriad of different formats, some of which are more important than others. In addition, file repositories outside of a user's personal computing system – including those in the web as well as those stored on other network users' computers – are frequently needed, increasing the number of accessible files significantly.

In order to be able to access and manage this huge information space, proper tools are needed. The user must be able to locate and get access to the desired piece of information rapidly and effortlessly, as well as get computational support for filtering, categorizing, organizing, and processing information.

One of the most important components in efficient content management is search: it provides means to locate the required information source and is hence needed before any further management can take place. In digital domain, search engines are used to retrieve the documents that fulfill the search query formulated by the user.

An interesting aspect in digital searching is collaboration. Information retrieval researchers have suggested that information seeking has always been a social process [Wilson, 1981]. Collaborative searching may concern search query generation, result browsing, or both. Romano et al. [Romano et al., 1999] have studied integrating information retrieval with group support systems. They point out that even though searching for relevant material is often vital to the progress of many groupware

---

[5] With kind permission of Springer Science and Business Media. © Springer-Verlag London Ltd. Lehikoinen J, Salminen I, Aaltonen A, Huuskonen P, Kaario J. Meta-searches in peer-to-peer networks. To appear in Personal and Ubiquitous Computing, 2006.

sessions, no support for collaborative searching is provided within groupware systems. Further, they combine the two paradigms and introduce Collaborative Information Retrieval Environment (CIRE).

One form of collaborative searching is collaborative filtering [Maltz & Ehrlich, 1995]. Collaborative filtering allows users among other things to annotate documents; the annotations can then be used as search criteria. In order to function properly, collaborative filtering requires a critical mass of users and annotations.

Liu et al. [Liu et al., 2002] proposed another approach to collaborative effort in search. Instead of using filtering, their approach is to allow users to express search goals in natural language and use "common sense" reasoning to translate the question into an effective query. They use the Open Mind knowledge base [Singh, 2002] for the source in reasoning. The Open Mind project allows a web-community to collaboratively build a knowledge base of simple facts, such as "A golden retriever is a kind of dog". Open Mind Common Sense project proves there is potential in using knowledge of a web-community in reasoning and information searches.

Peer-to-peer (P2P) networking is a potential, yet largely unexplored (especially from the user's point of view) technology for providing support for collaborative searching. Peer-to-peer networking can be defined as follows [Schollmeier, 2002]:

> *Distributed network architecture may be called a Peer-to-Peer (P-to-P, P2P,...) network, if the participants share a part of their own hardware resources (processing power, storage capacity, network link capacity, printers,...). These shared resources are necessary to provide the Service and content offered by the network (e.g. file sharing or shared workspaces for collaboration). They are accessible by other peers directly, without passing intermediary entities. The participants of such a network are thus resource (Service and content) providers as well as resource (Service and content) requestors (Servent-concept).*

As one can see, the point in P2P is sharing, i.e. both getting something from others while at the same time providing the others with something. Resources that can be shared include content, disk space, CPU cycles, or network capability. We aim at yet

another form of sharing – sharing personal information regarding the success of prior searches.

In this paper, we present an enhanced collaborative search mechanism for peer-to-peer networks. Our method takes P2P searching beyond mere individual search experience by allowing implicit, anonymous, and asynchronous collaborative searching. This is achieved by taking into account the searches made by other network users in the past, and creating a result relevancy indicator based on the previous searches.

The rest of the paper is organized as follows. First, we review the related work on the area, including peer-to-peer networks in general, and search techniques in P2P networks. We then present our own approach, including the concept, followed by the design description. Finally, the method is discussed and the paper concluded.

7.2 RELATED WORK

Even though definitions – such as the one above – for P2P exist, there is no clear consensus on the exact semantics of P2P. As Singh puts it [Singh, 2001]:

> *P2P can be defined most easily in terms of what it is not: the client-server model.*

On the one hand, P2P is seen to drive a major paradigm shift in distributed computing, implying numerous novel application areas. On the other hand, e.g. [Waters, 2001] states that P2P is not a novel technique; the term has been used for decades in somewhat different contexts. Nevertheless, in this paper we follow loosely the Schollmeier's definition given above.

Content sharing is the most common public embodiment of P2P. The most common Internet-based content sharing technologies include Gnutella and FastTrack. The basic idea is to allow any connected member to share his/her content to any other member, as well as to download shared content from others. For discussions on the upsides and downsides of P2P content sharing, including considerations for potential application areas beyond file sharing, see [Parameswaran et al., 2001]. For a concise comparison with some P2P projects, see [Wiley, 2001].

One of the most common pure P2P networks today is Gnutella. In order to connect to Gnutella network, the user needs a client application adhering to the Gnutella protocol. This application then connects to one of the several known hosts that are almost always available and have a permanent broadband Internet connection. These hosts, in turn, forward the port and IP addresses to other peers. The interaction between peers takes place by the means of messages. For more information on the Gnutella protocol, refer to [Gnutella]. For discussion on Gnutella network topology, refer to e.g. [Ripeanu, 2001].

Another concept used in Gnutella is the message time-to-live (TTL), or horizon [Oram, 2001, p. 110]. Each message in Gnutella contains a number describing how many hops between the peers it is allowed to do before it dies (6-7 by default). In some cases this prevents a certain node from reaching a node that would offer the content it requests.

An interesting feature in Gnutella networks is its ability to act as a very dynamic search engine. The Gnutella protocol allows each node to respond to a query whichever way it likes, therefore potentially allowing for many specialized search engines. A more advanced effort, InfraSearch (later purchased by Sun and renamed to JXTA), is a specialized search engine based on Gnutella protocol (http://search.jxta.org; for technical documentation, see http://search.jxta.org/JXTAsearch.pdf).

Information searching is an essential operation in content sharing; knowing how to find the desired content may not be trivial. Technically, searching in P2P is very different from traditional client-server searches. When searching using a search engine (e.g. Google), the search is performed against a huge index file maintained by numerous servers in the search engine site: it does not target real web pages and is therefore not real-time. P2P searches, on the contrary, are based on propagating the search query to the network via participating peers (until the TTL expires), which results in dynamic, nearly real-time answers and results that were found on the very computers where the information is stored. For more discussion on P2P searches, refer to e.g. [Miller, 2001, pp. 194-203].

Enhancing P2P search efficiency has been studied earlier. One such study concerns a distributed search service for mobile applications [Lindemann & Waldhorst, 2002]. The ´study introduces Passive Distributed Indexing (PDI), which is a search service targeted at file sharing applications. The focus is on the design of the service, as well as performance analysis.

Even though we are not aware of any research activities dealing with P2P collaborative searching, efforts towards P2P collaboration in general do exist. For example, Groove is a P2P application (http://www.groove.net), described as "a peer-empowering form of groupware"; it is also called peerware [Waters, 2001]. It is aimed at a P2P collaboration tool, for working in "secure mobile shared spaces". Groove provides group members with tools to collect all documents, messages and applications related to the group's activity. Everything is replicated to each member's computer(s), and is available for online or offline use. Groove may support activities such as shared real time editing.

An interesting concept impromptu collaboration [Kortuem et al., 2001]. In essence, it is proximity-based ad hoc interactions using mobile devices. The characteristics of impromptu collaboration are as follows:

- it is opportunistic: it allows people to take advantage of and make use of an opportunity that presents itself
- it is spontaneous: no prior human planning or preparation is required
- it is proximity-based: collaboration is made possible by physical proximity of two or more users
- it is transient: interactions are short-lived, seldom lasting more than a few minutes (or even seconds).

Kortuem et al. further present several usage scenarios concentrating on impromptu MP3 file sharing. They also analyze the differences between Internet and mobile PAN file sharing and point out issues in social, usage, and technical context.

7.3 OUR APPROACH

Searching for content is one of the most frequently performed tasks in P2P computing. One of the keys to success is formulating an effective search query. However, Spink et al. [Spink et al., 1999] found out that search engine users were quite unsure of how to formulate search queries that contain multiple search terms. The users were not certain if the words act as a phrase or if they are connected with a Boolean operator (such as OR, AND, or NOT). Therefore, it would be very useful to have support from the people who can generate efficient queries.

Since a P2P system enables computers to communicate with each other, it also implies that the users of these computers can collaborate via this system. Therefore, we decided to study how collaboration could be brought into P2P searching process. The need for collaboration is emphasized by the fact that computers in P2P system are often geographically distributed, which means that the users do not have the support of co-located peers. Also, other forms of collaboration over distance than communication via media channel (such as chat or audio) need to be discovered since people often face problems caused by the limited social interaction [Mark et al., 2003].

Our approach is based on augmenting the received search result set with query-related metadata automatically. For every search result set that the user receives for his/her query, the user's interaction with this set is tracked and stored as metadata. These metadata reflect the importance and the relevancy of a single item to the creator of the query. The augmented results try to answer the question "How well does a certain result item satisfy the query the user has just entered?" These augmented result sets can then be used for enhancing the user's subsequent queries, as well as for other peers' future queries. Naturally, in some cases a more direct peer support might be beneficial. For example, the user could start a chat session with or send an instant message to the peer who has provided some piece of relevant information.

In PDI [Lindemann & Waldhorst, 2002], the results from queries, i.e. indices of found files, are stored in local caches. In our approach, we store the actual queries and their relevancy or "goodness" for producing desired outcome.

7.3.1 Concept

On a conceptual level, the meta-search works as follows. When a user performs an enhanced peer-to-peer network search, s/he types in the keywords needed to obtain desired results as usual. Once the results are retrieved, they are presented to the user (as usual). In addition to traditional results listing containing only the matching documents or links, this enhanced list also displays relevance information for each link. In other words, it conveys how the obtained results have been used in the past queries by other users. This information is sent along with the regular result listing. It can be used in judging whether a link is relevant or not. An important aspect is that the relevancy is always based on a particular query instead of a particular file. Figure 1 depicts the overall functionality of meta-search.

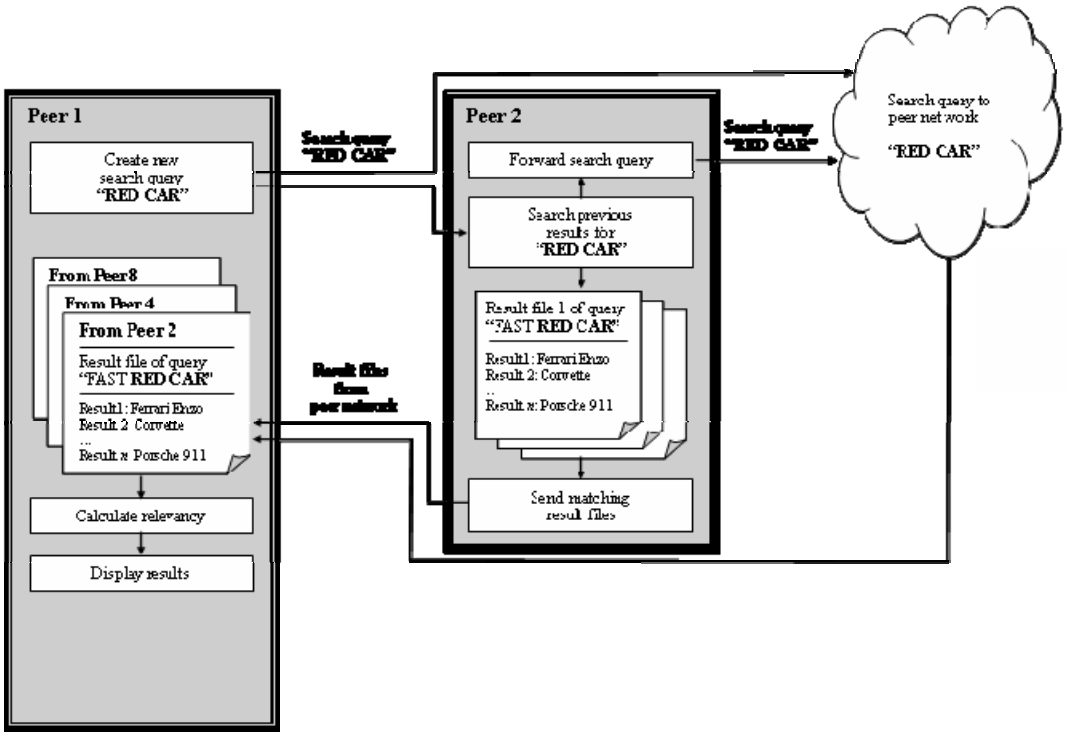

Figure 1. The overall schematics of the meta-search method.

The very nature of the P2P query propagation makes it possible to enhance the search query as described above. Since the query travels through a very large number of peers, it can be effortlessly augmented with relevance information collected along the route. The downside, obviously, is increased network traffic.

The following imaginary example further clarifies the concept:

"Bob types a query "depeche mode, frame" on his computer and starts the search and the query is then sent to the peer-to-peer network. The search software in each peer receives the query, and checks whether the content stored in the local computer satisfies the search criteria. One of these computers belongs to Alice and her computer contains an image file titled "Depeche Mode – A broken frame.jpg". Since this file satisfies the query, her computer sends a link to this file back to Bob. Further, Alice has herself earlier typed a similar query, which resulted in many items on different peer computers. Of these items, Alice has downloaded the image from Charles. Now, Alice's search software sends also this information back to Bob's computer (i.e., the address of Charle's computer as well as other query-related metadata). After sending the results, Alice's computer propagates the query to her peers. As the query travels along in the network, Bob starts to receive results. These results are links to the actual resources where the match for the query is found, augmented with search-related metadata of how they were obtained. Bob notices that Alice's terminal has a matching file; he also sees that Alice has downloaded the image from Charles. Therefore, this is a potential link to follow. More results come in, and Bob notices that many others have also downloaded from Charles. He then concludes that it might be a relevant source, and starts downloading himself."

Our approach enhances the searching process and provides at least the following benefits:

- it provides peer support automatically without explicitly asking their help. The user is able to estimate the relevancy and quality of a file based on the augmented metadata of previous queries;

- it saves the previous searches, thus collecting a search history. Collecting search history has been pointed out as an important aspect by previous studies[Spink et al., 1999; Spink et al., 2001]; and

- it helps sharing knowledge within peers as well as makes it manageable by making relevant information retrievable. In addition, the method allows people with knowledge to be accessed.

Browsing previously made searches adds a learning aspect to the searching since the user is able to view search queries that have produced good results. This may help the user to construct similar queries and/or search strategies. Accessing peers' saved searches expands the process to collaborative learning. A potential downside of the method is that the users may feel that their privacy is compromised when their searching behavior is revealed.

7.4 DESIGN

There are two major components related to meta-search: matching similar searches, and calculating the relevancy value for each match. Further, we need to specify the relevancy file format. We will next describe each of these components in detail.

7.4.1 Calculating the Relevancy Value

The relevancy value is a single integer number that describes how well a previously performed search result item matches the current search query. In order to calculate the relevancy, user interaction with the previous query and result set must be stored. There are two potential approaches:

A1. Track the user interactions during the query and download.

A2. In addition to A1, track also user interactions after downloading (e.g., how many times a document has been opened, how long has been spent reading/viewing/listening to it, and so forth).

The approach A2 is much more powerful and provides detailed relevancy information on the result item; however, it requires an extensive system that tracks the file and the applications that can be used to access the file. As a consequence, we decided to take the approach A1 in the first phase.

What, then, are relevant metadata to collect? In our present design, at least the following interactions with each item of any search result set are tracked. The list below is not exhaustive, but merely an indication of potentially useful pieces of information.

Indicator		Description	Quantification
File exists		If a file matching the search criteria in relevancy file is found on the peer, it usually means that the search was successful	Discrete function with two points in the range. 1 if file exists, 0 if it does not. No negative values.
File downloaded		Even if there is no matching file on the peer, the P2P client can tell that some file was actually downloaded but later deleted. This is a weaker indicator having less weight than the previous but still positive indicator for successful search.	Discrete function with two points in the range. 1 if file was downloaded, 0 if not. No negative values.
Timeline	Time between the last access and the received query	If a file matching the received query is accessed close to the time when the query is made, it probably indicates that the file is still considered relevant.	A decreasing function of time with form f(t) = 1/t, where f(t) is always between]0,1]. No negative values.
	Time between last access and the download	The longer the time between the download time and last access data, usually indicates good match since the file is still considered relevant.	An increasing function of time with form f(t)=sqrt(t), where f(t) is always between [0, 1]. No negative values.
	Time between download and the creation date	The creation date of the file may be available from the metadata included in the file. The shorter the time between the download and creations, the fresher the file is. This freshness of the file is naturally a subjective indicator	A decreasing function of time with form f(t) = 1/t, where f(t) is always between]0,1]. No negative values.
Integrity		Any data that is available from the network. For example, Kazaa and Gnutella offer integrity values.	Discrete function where integrity value is mapped to the range [-1, 1]
Preview		During the download, it is possible to preview the	Discrete multipart function where f(n) = -1 if file has been

	file. If the file has been previewed and then disregarded, that is clear indication of failed search. On the other hand if after the preview the download is successfully finished, the preview can be used as a positive indicator	previewed and then cancelled, 0 if no preview, and then a linear function of *n* with values between]0,1] if positive number of previews with no cancel. *n* is the number of previews.
Subjective relevance	The P2P client can ask the user how well the query succeeded and this feedback can be used as a subjective indicator.	Discrete function from the user input that has a single value between [-1,1]
Number of results	The number of results the query returned.	Discrete linear function mapped to the range [-1,1]
Download aborted	Indicates whether the user has aborted already started download of the file matching the query.	Discrete function with two values in the range. –1 if download has been aborted, 0 otherwise.
Download paused	Indicates whether the file matching the query was started to download but the user paused the download to continue later.	Discrete decreasing function f(p) gets values from [1,0]. p is number of previews. No negative values.
Search more chosen	Indicates whether the user chose the search more option available in some P2P clients.	Discrete function with 0 if search more is not chosen and 1 if it was.
Query aborted	Indicates whether the user aborted the whole query before it was finished.	Decreasing function of time in form f(t) = $1/t - 1$, where t is positive amount of time from starting the query and where f(t) is always between [-1, 0]

Each item has a function that results in a real number between [-1,1]. In the table, the functions have not been strictly defined but only their form or shape is given. It is left to future study to find out what are actually the best functions to use.

For each file matching the query in relevancy file, a vector of interaction values can be now created on the receiving peer. This vector is returned with query results to the

query originator, which then further quantifies the vector to a scalar value calculating the dot product with a weight vector.

The values in the weight vector are also real values in the range [-1,1] but these values are subjective by their nature; i.e. every user has his or her preferences how to weight the interaction items. Therefore, it is important that the client gives the user a possibility to give his or her own values, although a good set of initial default values should also be available. Putting negative weight value for an interaction item actually transposes the meaning of the value function, i.e., the interaction item that usually would indicate a good query, weakens the query with negative weight value.

So, for each file matching the query, there is an index number between [-n, n] where n is the number of interaction items used to calculate it, 13 in the above table. The index indicates how good the query was related to the results. Negative values indicate poor quality, while positive values indicate a good quality query. To make things easier, a maximum index from all matching files is chosen to represent the whole query and that query index is then used to rank all queries in the relevancy file. These queries can be ordered according to the rank.

7.4.2 Query Matching Algorithms

When a query from outside arrives into a terminal, the recipient must check if there already exist results from previous queries that match the new query. In order to do that we need to create a way to compare two queries to see if the results returned from one query are relevant to another. If there is a common result set between two queries we can say that there is a match between the queries and they are relevant to each other.

A rough initial test for matching is performed with simple string comparison. For each search term in the received query, we compare each term in each previous query. Any previous queries are discarded from further study if they do now share any terms with the received query.

We then test the set of previous queries that contained matching strings for logical relevance with the received query. We use either or both the following algorithms: the Tree-levelling algorithm or the Minterm algorithm.

7.4.2.1 The Tree-levelling Algorithm

In simple cases the subset relation can be used to make the match. If query A is a subset of query B then its results can be directly used for query B. The Boolean operator AND requires that all parts of AND operation must match. Therefore a query with an AND operation is a subset of all the queries that can be formed by taking the parts alone.

Likewise, the OR operation extends queries. To match it is sufficient that any of the search terms in an OR query matches. Therefore a query without OR is a subset of the query with OR. Putting this together, we can match queries based on search terms and boolean operators. If search terms match, then queries with OR have the largest result set and all queries with AND or without any Boolean operator can be used as a common query. Similarly, if received query does not contain any operator, AND queries can be used as a common query.

Problems arise when the received query contains both AND and OR operators possibly ordered with parenthesis. For example, if there are two queries: Query A is “(Depeche AND Mode) OR (Broken AND Frame)” and query B is “Depeche OR Frame”. In these cases, we need to determine whether the results from query A are included in the results of the query B or vice versa.

In order to make comparisons between these queries, they must be first reordered and made structurally equivalent. All queries form tree structures where there is a Boolean operator as a root and search terms or other Boolean operators as children. In the leaf nodes, there is always a search term. In cases with no Boolean operators, an implicit OR node is assumed between the search terms. To make queries easier to compare, the query that has less nested operations, i.e. it’s query tree is lower, is transformed to equal height with the compared query tree. An OR-node is added as a parent to the transformed query and the original query is the left child of the new root node. The left child is then copied to the right side of the root. This new query has exactly the same semantic meaning as the original query. If needed the operation can be repeated until both queries have the same structure.

Ordering the queries starts by creating the query tree. In the previous example, the queries form trees as shown in Figure 2.

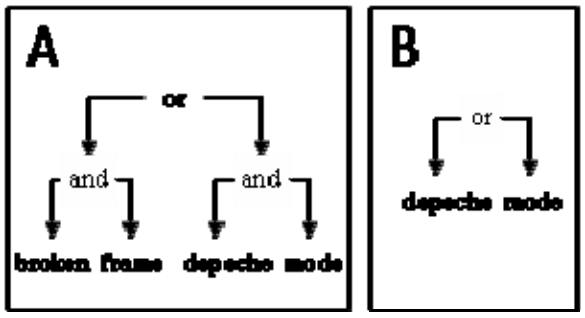

Figure 2. Two query trees that are to be compared.

In the example above, after creating a query tree, all search terms are arranged into alphabetical order. It is also seen that both queries start with an OR -query and the A-query on the left contains two AND queries as sub-queries for the OR query. The B-query on the right has no boolean operators under OR query. Since search terms on the B-query are also in the AND terms of the A-query, the results from the A-query are also relevant to the B-query. The following algorithm can be deduced.

1. Let B be the received query and A the query in the relevancy file
2. Create an ordered query tree if the search engine has not already done so.

    3.1. Order the search terms into alphabetical order.

    3.2. Make both queries equal in height by adding implicit OR nodes as parents to nodes in the lower query tree and copying the query as a right child of the new implicit OR node.

    3.3. Repeat 2.2 as many times as needed to make both trees equal in height. (Note that the lower tree will be a complete binary tree with many unneeded nodes.)

3. Start with the root node and compare the queries.

    3.4. If the B-query has an AND relation then also A must have an AND relation and in order for the queries to match, both left and right subtrees of B must also match to their counterparts in A.

    3.5. If the B-query has an OR relation, then it is sufficient that either of its subtrees match to its counterparts in A.

4. Two subtrees match if the subtrees are leaf nodes and they have the same search term. If they are not leaf nodes, then they are compared recursively like in step 3.

5. If the root node matches, then the query in relevancy file can be used for the query B.

In the example above, we need to add an OR node in the query B (see Figure 3). Since B has an OR node as a root, the whole query matches if either of the subtrees match. Next the left and right subtrees of the root node of B are checked if they match with the respective left and right subtrees of the root of A.

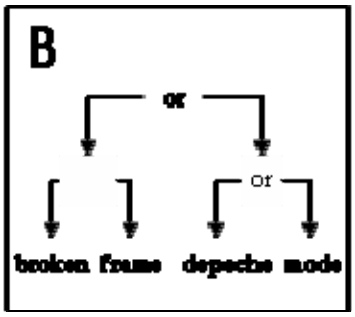

Figure 3. Adding an OR-node in the query B.

B has OR nodes so again, it is sufficient to see if either of the subtrees match. At the next level, there are only search terms, so in order to match, both queries must have the same search terms. In A the first search term is “broken” and in B it is “depeche” so there is no match. The right subtrees have terms “frame” in A and “frame” in B so these terms match. That means that the OR parent of the search terms in B also matches. Again this means that the whole B-query matches with A and A is therefore relevant to query B with regard to the term “frame”.

In implementation, when making both trees to equal height and structure, there is no need to actually create trees with extra nodes. Only those that are actually used can be copied using pointers or other similar methods that require less memory.

7.4.2.2 The Minterm Algorithm

This algorithm is a direct adaptation of the well known boolean algorithms used for optimising digital logic circuits, such as the Quine-McCluskey method [McCluskey,

1986]. However, the aim here is not to find minimal representations of query functions but to detect their equivalence.

Let the received query be denoted with B and a previous query denoted with A. The search terms contained in B are denoted with (B_1, B_2, … B_M) and the terms in A with (A_1, A_2, … A_N). We consider A and B as Boolean functions over the space of their possible binary input values, or $f_A(A_1, A_2, \ldots A_N)$ and $f_B(B_1, B_2, \ldots B_N)$. A true value '1' (one) of a variable denotes that the corresponding term results in a match among files of the peer. A false '0' (zero) denotes that there are no matching files.

The union of the both functions is then f = (A_1, A_2, … A_N, B_1, B_2, … B_M). We create for the union a truth table with 2^{N+M} columns and K rows, where K is $N+M$+the sum of the number of subtrees in A and B. The columns list the minterm expansions (sum-of-products, or possible logical combinations of the products of input variables or their complements) of f.

We then list all the variables in A and B in the rows, and fill the truth table cells with '1' where the minterm contain the true variable, and '0' otherwise.

Next we traverse the query trees A and B (as in the Tree-levelling algorithm):

For each subtree in A and B, create a new row, and fill the row cells recursively as following:

> For each subtree that has an AND operator for its subtrees, a cell in the row is '1' if all of the subtrees have '1' in their corresponding cells (recurse).
>
> For each subtree that has an OR operator for its subtrees, a cell in the row is '1' if any of the subtrees have '1' in their corresponding cells (recurse).
>
> For each subtree that consists of just one variable, a cell in the row is '1' for cells where the corresponding minterm is true for the variable, and '0' otherwise (end of recursion).

The resulting truth table has rows for f_A and f_B. The queries are equal if the rows are equal. Query A is a superset of B if an AND operation between rows A and B = B.

This algorithm suffers from combinatorial explosion with large M and N, but in practice usual queries are rather small, and the resulting truth table will then remain relatively compact.

7.4.3 The Relevancy File

An essential component of the meta-search method is the relevancy file. This is an XML (eXtended Markup Language) file containing information on both the query and the results. When the user types in a query, the file is created, containing only the first section (see below for an example file). This file is then sent to the peers seen by the user. The peers check whether a match is found.

When a query arrives and a match is found, the original sender is replied with information on the match (this is called a direct match). In addition to its local files, a peer may also scan through its own previous relevancy files. If any of them contains a subset of the requested query, the relevancy file in question is also transmitted. The links contained in these files are called indirect matches. Finally, the peer forwards the original relevancy file to the peers it can see (excluding the one that it received the file from).

Once the sender starts receiving direct and indirect matches, it adds each direct match to the current relevancy file (unless it is already included), and collects information from the indirect matches. For each link found in BOTH direct and indirect matches, it updates the current relevancy file to contain the metadata information from indirect matches. Thus, only those resulting links that satisfy the original query are included in the results by default. However, the user is also able to view the links that are not included in direct matches by manually choosing to include these results in the list.

Once the search is finished, the relevancy file may look like the following:

```
<?xml version="1.0"?>
<relevancy xmlns="http://www.organization.org/namespaces/meta-search">
  <query>
     <sender-ip>xxx.xxx.xxx.xxx</sender-ip>
     <query-string>depeche and mode</query-string>
     <search-started>2003-12-08 16:16:29.23</search-started>
     <search-finished>2003-12-08 16:19:12.01</search-finsihed>
     <invalid-after>2003-12-15 16:16:29.23</invalid-after>
     <time-to-live>10</time-to-live>
     <search-more>true</search-more>
     <search-aborted>false</search-aborted>
     <user-rating>7</user-rating>
     <results>
        <result>
           <host-ip>xxx.xxx.xxx.xxx</host-ip>
           <nickname>nick name</nickname>
           <bandwidth>512</bandwidth>
           <number-of-results>1</number-of-results>
           <files>
              <filename>broken frame</filename>
              <integrity>4</integrity>
              <events>
                 <download-started>2003-12-08 16:17:32.45</download-started>
                 <previewed>2003-12-08 16:18:43.89</previewed>
                 <previewed>2003-12-08 16:20:01.67</previewed>
                 <download-paused>2003-12-08 16:20:23.04</download-paused>
                 <download-started>2003-12-08 16:23:12.31</download-started>
                 <download-finished>2003-12-08 16:27:22.19</download-finished>
                 <others-paused>yes</others-paused>
              </events>
           </files>
        </result>
     </results>
   </query>
</relevancy>
```

Once the user starts interacting with the result set, another relevancy file is created. This file is identical to the one created earlier, with the exception that it only contains the links the user him/herself has visited, and the visiting information, visiting time, and downloaded content by this user only. Once the searching session is terminated, the original relevancy file is destroyed and the file containing only the user's own interactions is kept. This is necessary to avoid cumulating the relevancy information.

7.4.4 Implementation Issues

We have developed our own networked application development platform called Message Exchanger or MEX. It was originally designed for wearable computing but can easily extended to generic network application design. The MEX is described in

detail in [Lehikoinen et al., 1999]. The MEX architecture is designed to be flat and modular without distinction between client and server applications, any application can either use or provide services to other applications. A concept of application is, actually, a collection of services provided or requested through the Message Exchange. The MEX enables inserting and removing components dynamically and it allows them to communicate with each other even if they don't have any previous knowledge of each other. The key idea behind MEX is anonymous services in the sense that they don't know who is going to use the service they provide nor do they care where the information they receive came from.

The original MEX does not have pure P2P architecture but we have further developed MEX to pure P2P system called MexNet. MexNet has been implemented and used for usability tests in [Suomela et al., 2001].

However, our meta-search design does not require MexNet or any particular P2P infrastructure to work. It has been designed to be implementable on any system, only requirements are the system allows making queries and returning data back to query originator. Obviously, all file sharing networks fulfill these requirements.

On MexNet, searching files and returning relevancy vectors can be implemented as seen on figure 4.

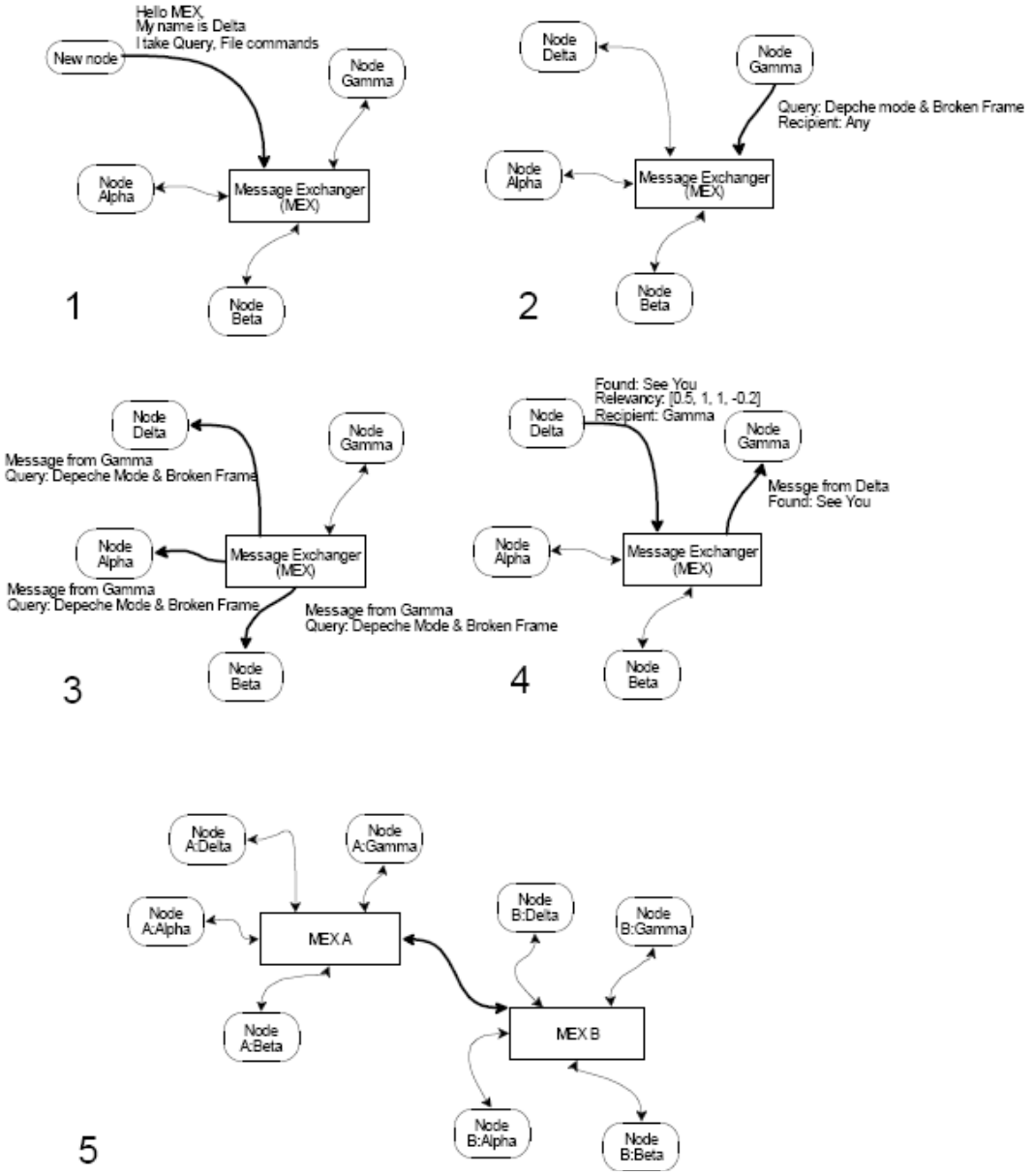

Figure 4. Implementing file search on MEX and MexNet

Since MexNet is fairly simple extension to MEX to form P2P network of MEX nodes, the meta-search queries are here presented as they should occur on single MEX node.

Each MEX node may have one or more clients attached to it. MEX takes commands from clients and sends them to other clients that have claimed interest on those commands. In part 1 of figure 4, a new client attached itself to MEX node. It introduces itself to MEX and tells what commands are in its interest. In part 2 and 3, one node sends a query into network. It does not specify any particular recipient, so MEX distributes the query to all its clients that have announced interest in receiving

queries. In part 4, one client returns its own results for the query with the relevancy vector. Similarly, all other nodes would return results of the query and all relevancy vectors they may have.

The part 5 of the figure 4 shows how in MexNet, the MEX nodes can form P2P network. In MexNet, queries and their results are distributed throughout the whole network.

7.5 DISCUSSION & FUTURE WORK

Sending the interaction values of the user in the peer receiving the query back to the originating peer reveals some personal information. Naturally, this disclosure of potentially private information must be made clear to anybody using the system and agreeing to help others in their searches. One way to mitigate the problem is to send the weight information with the query and to do the computation on the receiving peer, and then only aggregate information is sent back. This method consumes computing power on every machine the query reaches and therefore uses resources of those who help instead of the resources of the person who asks for help. Even doing the relevancy calculations on the receiving query does not totally prevent a determined attacker to get details. He or she could send the same query several times and every time use a different weight vector with one non-zero value and all other weights as zero. Then combining the results will produce the original relevancy values.

An inherent feature of P2P networks is its real-time nature, i.e. search results reflect the current state of the peers. Therefore, a peer referenced to in a previously stored match may not be online when the relevancy is used. In most cases, this is not the problem – P2P networks are known to be very dynamic as peers come and go, and it is normal behavior for a download to be interrupted and then continued again.

One attractive feature of the meta-search design described in this paper is that it is relatively effortless to implement and integrate into existing P2P clients. The current design only uses information internal to the client (i.e., interactions with it), and the files in the shared folder(s). Further, since relevancy files are treated as any other

files, they do not interfere with non-enhanced clients. On the contrary, relevancy files can be downloaded as any other files on the shared folder(s).

Probably the most challenging issue with meta-search deals with privacy. Users are not willing to reveal any additional information that is not considered necessary. Even though the users of meta-search enabled clients must be allowed to enable and disable the feature at will, this remains an open issue. It should be noted, however, that the users of P2P systems are already used to the concept of allowing unknown peers access to their very own computer hard disks.

A potential approach to decrease the fear of compromized privacy is to allow the users to define public and private interaction metadata elements; only the public information would be transferred when a query is received.

A promising approach is to extend meta-search beyond using only information internal to the clients. However, this increases the complexity of the implementation, as well as the risk of compromized privacy. This extension should be designed with great care.

One problem in the peer-to-peer network queries will be redundancy in search and integrity of the search results. The redundancy can be tackled with relevancy file rather effectively, like presented earlier. However, users are likely to use different ranking of value in their metadata. This creates challenges to integrity and the relevancy value will be compromized. Thus, a successful search will need more redundancy in queries to statistically determine the relevancy value. One possible solution to this is the use of common sense knowledge, such as Open Mind Common Sense [Singh, 2002]. The common sense knowledge would be used to determine the value proposition of the metadata, for example to state: "Bob always listens to newly downloaded songs within 24 hours." and "Alice probably listens to new songs within one week." In this example both of the users value the music as much, so the relevancy value is equal with very different numeric time value .As the data resides on a web-community knowledge base, the query needs only a binary value whether two peers in the network are using same knowledge for values of their metadata. This topic needs obviously more research, but it has potential of utilizing effectively the social nature of peer-to-peer networks.

The next step in developing meta-search further is to implement a prototype version and experiment with issues such as traffic increase and the most potential attributes to be tracked and functions to use with them.

7.6 CONCLUSION

We have designed an asynchronous, implicit collaborative search method for peer-to-peer networks, called meta-search. The purpose of meta-search is to make searching more efficient by exploiting previous queries made by other peers. This is done by tracking the user's interaction with a search query and result set. When a peer then receives a query, it not only checks whether it has any matching files on share, but it also checks its previous queries to see if it has performed similar queries in the past. It then informs the requesting peer on the success of those previous searches by providing a relevancy value for each match. Meta-search consists of three main components: the relevancy calculation algorithm; the query matching algorithm; and the relevancy file format.